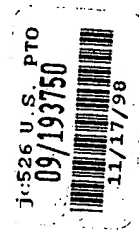


do not
modify
~~to be changed~~
YR

Appendix



5. Object/Structure

5.1 UI Controller Interface Objects

5.1.1 LIBCLASS CdpgToUIC

to provide communication from the Damage Page to the UIC

HUIC hUIC;

CdpgToUIC() { hUIC = 0; }

HUIC GethUIC() { return hUIC; }

HUIC SethUIC(HUIC h) { return hUIC = h; }

// set up window extnal data to communicate with UIC

void SetWindowData(HWND hwnd, void FAR *ptr); // set ptr

void FAR * GetModulePtr(HWND hwnd); // get ptr

// send information to other loadable modules through UIC

void TriggerEvent(DWORD modType, UINT ulID, WPARAM wP, LPARAM lP);

LRESULT SendMessage(DWORD modType, UINT msgID, WPARAM wParam, LPARAM lParam);

BOOL SendAction(DWORD modType, UINT actionID, WPARAM wParam, LPARAM lParam);

BOOL SendEvent(DWORD modType, UINT eventID, WPARAM wParam, LPARAM lParam);

DWORD QueryModule(DWORD dwIndex);

5.1.2 CdpGFromUIC

to provide communication from the UIC to the Damage Page.

```
void dll_InitializeDLL(HANDLE, WORD, WORD, LPSTR)
    initialize the DLL,
    register a custom edit control class for grid in part edit module
```

```
void dll_UnloadDLL(int)
    do nothing
```

```
BOOL dll_RegisterUICInstance(HUIC)
    if already registered
        return FALSE // to prevent more than one instance
    else
        save UIC
        return TRUE
```

```
void dll_UnRegisterUICInstance(HUIC)
    clear handler of UIC, hUIC
```

```
BOOL dll_HandleAction(HUIC, DWORD, UINT, WPARAM, LPARAM)
    if ( broadcast message or message for this module)
        return dpgMgr.DoMessage(uiID, wParam, lParam)
    else
        return FALSE
```

```
LRESULT dll_HandleMessage(...)
    if (broadcast message or message for this module)
        return dpgMgr.DoMessage(uiID, wParam, lParam)
    else
        return FALSE
```

```
BOOL dll_HandleEvent(...)
    if (not broadcast message or message not for this module)
        return FALSE
    else switch (message)
        case ADP_OPEN
            set cursor to WAIT cursor
            open data base
            set cursor back to normal
        case ADP_CLOSE
            close database and user interface windows
        case ADP_SHOW
            open/show user interface windows
        default
            pass message to the manager dpgMgr.DoMessage
```

```
DWORD dll_QueryModule(HUIC, DWORD)
    do nothing
```

```
int dll_TranslateMessage(HUIC, LPMSG)
    call TranslateMessage of dpgMgr to translate message
```

5.2 Manager and User Interface Objects

5.2.1 LIBCLASS CdpgWindow

```
HWND hWnd;  
virtual ATOM Register(LPSTR lpClass);  
virtual HWND OpenWindow(LPSTR lpClass, HWND hwndParent,  
LPRECT lpRect, DWORD dwStyle = 0, int nChildID = 0);  
virtual HWND OpenModelessDialog(LPSTR lpClass, HWND hwndParent);  
virtual HWND OpenModalDialog(LPSTR lpClass, HWND hwndParent);  
  
static CdpgWindow * GetThisFromWindow(HWND hwnd);  
  
CdpgWindow() { hWnd = 0; }  
virtual ~CdpgWindow() {}  
  
void SetHwnd(HWND hwnd) { hWnd = hwnd; }  
HWND GetHwnd() { return hWnd; }  
  
virtual HWND Open(HWND hwndParent, LPRECT lpRect = NULL) = 0;  
  
virtual void BeforeWMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);  
virtual LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);  
virtual LRESULT WMClose(WPARAM wParam, LPARAM lParam);  
virtual LRESULT WMDestroy(WPARAM wParam, LPARAM lParam);  
virtual LRESULT WMMouse(UINT uMsg, WPARAM wParam, LPARAM lParam) { return  
FALSE; }  
virtual LRESULT WMKbd(UINT uMsg, WPARAM wParam, LPARAM lParam) { return  
FALSE; }  
virtual LRESULT WMCtlcolor(WPARAM wParam, LPARAM lParam);  
virtual LRESULT WMSize(WPARAM wParam, LPARAM lParam) { return FALSE; }  
virtual LRESULT WMPaint(WPARAM wParam, LPARAM lParam) { return FALSE; }
```

```

virtual LRESULT WMScroll(UINT uMsg, WPARAM wParam, LPARAM lParam) { return
FALSE; }
virtual LRESULT WMDpgMsg(WPARAM wParam, LPARAM lParam) { return FALSE; }
virtual LRESULT WMCommand(WPARAM wParam, LPARAM lParam) { return FALSE; }
virtual LRESULT WMAdpMsg(UINT uMsg, WPARAM wParam, LPARAM lParam) { return
FALSE; }
virtual LRESULT WMOthers(UINT uMsg, WPARAM wParam, LPARAM lParam) { return
FALSE; }

LRESULT DoMessage(UINT uMsg, WPARAM wParam, LPARAM lParam)
{ return SendMessage(hWnd, uMsg, wParam, lParam); }

```

virtual ATOM Register(LPSTR lpClass)

lpClass : class name string

Returns: the ATOM that uniquely identifies the registered class

Description: register a window class

```

    if class exists
        return OK
    else
        set up the WNDCLASS data structure
        return RegisterClass(&wsc)

```

virtual HWND OpenWindow(LPSTR lpClass, HWND hwndParent,
LPCRECT lpRect, DWORD dwStyle, int nChildID)

lpClass: class name string

hwndParent: parent window handler

lpRect: far pointer of rect area

dwStyle: window style

nChildID: child window control ID

Returns: window handler

Description: create a window as specified by the input parameters and return the
window handler.

```

    if (window not exists)
        if (register failed)
            return FALSE
        else
            CreateWindow
    else
        ShowWindow in its previous position
        return the handler of window

```

virtual HWND OpenModelessDialog(LPSTR lpClass, HWND hwndParent)

lpClass: window class name string

hwndParent: parent window handler

Returns: window handler

Description: Create a modeless dialog and return its handler.

if (window not exists)

{

pass dialog processing procedure dpgDialogProc

CreateDialog

}

ShowWindow in its previous position and appearance

return the handler of window

virtual HWND OpenModalDialog(LPSTR lpClass, HWND hwndParent)

lpClass: window class string

hwndParent: parent window handler

Returns: window handler

Description:

make procedure instance

Create the dialog box with procedure dpgDialogProc

free procedure instance

return status of the dialog creation

static CdpWindow * GetThisFromWindow(HWND hwnd)

hwnd: handler of window

Returns: object pointer

Description: get object pointer from the window property list

virtual LRESULT WMCtlcolor(WPARAM wParam, LPARAM lParam)

wParam: Handler of Device Context(HDC) of the control child

lParam: HIWORD(lParam) = control type

LOWORD(lParam) = child window handler

Returns: handler of the brush

Description: handle the WM_CTLCOLOR for dialog box colors

switch(HIWORD(lParam))

case CTL:COLOR_DLG:

change dialog background

return with color of the dialog background

case CTLCOLOR_STATIC:

case CTLCOLOR_BTN:

change text color and set background mode to transparent

return with the background color

case CTLCOLOR_LISTBOX:

case CTLCOLOR_EDIT:

set text color and set background mode to transparent

return with white brush for the rest area

Free functions:

```

//*****
// general message processing procedure for window
//
LRESULT dpgGeneralProc(CdpgWindow * pcWnd, IIWND hWnd, UINT
                      uMsg, WPARAM wParam, LPARAM lParam)
pcWnd: pointer of the object
hWnd: handler of window
uMsg: window message
wParam: window message word size parameter
lParam: window message long size parameter
Returns: the processing return from calling message processing procedure
Description:
switch (message type)
    case WM_CLOSE
        call WMClose of the object to process
    case WM_DESTROY
        call WMDestroy of the object to process
        RemoveProp
    case WM_MOUSEMOVE
    case WM_LBUTTONDOWN
    case WM_LBUTTONUP
    case WM_LBUTTONDBLCLK
        call WMMouse of the object to process the message

    case WM_CHAR
    case WM_KEYDOWN
        call WMKbd of the object to process the message
    case WM_VSCROLL
    case WM_HSCROLL
        call WMScroll of the object to process the message
    case WM_SIZE
        call WMSize of the object to process the message
    case WM_PAINT
        call WMPaint of the object to process the message
    case WM_DPG_MSG
        internal messages of the damage page
        call WMDpgMsg of the object to process the message
    case WM_COMMAND
        call WMCommand of the object to process the message
    default
        if (Audapnt internal message, sent it to WMApMsg)
            return TRUE
        else
            WMOthers of the object to process the message
}

```

```

//*****
// processing message procedure for window
//
CEXTERN LRESULT WINAPI dpgWindowProc(HWND hwnd, UINT
    uMsg, WPARAM wParam, LPARAM lParam)
hwnd: window handler
uMsg: window message
wParam: window message word parameter
lParam: window message long parameter
Returns: the return value of the calling window procedure
Description:
    get the window pointer for GetThisFromWindow
    switch (message) {
    case WM_CREATE
        link object to the window
        Set the window pointer to its property list
        pass the message to the WMCreate procedure of the object
    default
        if the window pointer is valid
            if dpgGeneralProc
                return NULL
    }
    // call default window procedure
    return DefWindowProc()

//*****
// processing message procedure for dialog
//
CEXTERN LRESULT WINAPI dpgDialogProc(HWND, UINT, WPARAM,
    LPARAM)
hwnd: dialog handler
uMsg: dialog message
wParam: window message word parameter
lParam: window message long parameter
Returns: the return value of the calling dialog procedure
Description:
    get the window pointer from GetThisFromWindow
    switch (message) {
    case WM_INITDIALOG
        link object to the window
        Set the window pointer to its property list
        call BeforeWMCreate to set font
        pass the message to the WMCreate procedure of the object
    case WM_CTLCOLOR
        if (window pointer is valid)
            call WMCtlcolor to set the control color
    default
        if the window pointer is valid
            if dpgGeneralProc
                return TRUE;
    }
    // message not processed
    return FALSE

```

5.2.2 LIBCLASS CdpMgr:public CdpWindow

Class for the Damage Page Manager, response for the creation of child window objects, synchronization, territory decision and internal state maintenance.

```
HACCEL hAccel;    // accelerator
HWND hwndChild[childIndexMAX]; // children
CHILDINDEX nCurrentChild; // current child
int damageState;
    // user selected states, either damageOff or damageOn
int drillState;
    // user selected states, either drillOff or drillOn
int naviState;
    // user selected states, either naviNoIcon or naviYesIcon
int plistState;
    // user selected states, one of the three plistGraphicOnly, plistPartlistOnly,
    // or plistboth

int npwTotal;    // width in pixel of total
int npwPlist;    // width in pixels of part list
int npwGW;       // width in pixels of graphic window
int nphTotal;    // height in pixels of total
int nphDamage;   // height in pixels of damage list
int nphNavi;     // height in pixels of navigator ICON
int nphPlistMargin; // height in pixels of part list margin

-----
CdpMgr();

int GetDamageState() { return damageState; }
int GetDrillState() { return drillState; }
int GetNaviState() { return naviState; }
int GetPlistState() { return plistState; }
int GetNpwTotal() { return npwTotal; }
int GetNphDamage() { return nphDamage; }

HWND Open(HWND hwndParent, LPRECT lpRect);

LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
LRESULT WMClose(WPARAM wParam, LPARAM lParam);
LRESULT WMDestroy(WPARAM wParam, LPARAM lParam);
LRESULT WMKbd(UINT uMsg, WPARAM wParam, LPARAM lParam);
LRESULT WMDpgMsg(WPARAM wParam, LPARAM lParam);
LRESULT WMCommand(WPARAM wParam, LPARAM lParam);
LRESULT WMApMsg(UINT uMsg, WPARAM wParam, LPARAM lParam);
LRESULT WMOthers(UINT uMsg, WPARAM wParam, LPARAM lParam);

int TranslateMessage(LPMSG lpMsg);
LRESULT AnnounceMessage(UINT uMsg, WPARAM wParam, LPARAM lParam);
```

void MoveChildren();

Returns: nothing

Description:

```
Show the Damage Page manager window
Show the three-line damage list
If (naviState is naviNoIcon)
{
    Hide window of Navigator with ICON
    Show window of Navigator without ICON
}
else
{
    Hide window of Navigator without ICON
    Show window of Navigator with ICON
}
switch (Part list state)
{
case Part List Graphic Window Only:
    Hide Child Window of Single Part List
    Hide Child Window of Multiple Part List
    Show Child Window of Graphic Window
case Part List Window Only
    Hide Child Window of Graphic
    Hide Child Window of Single Part List
    Show Child Window of Multiple Part List
case Part List and Graphic
    Hide Child Window of Multiple Part List
    Show Child Window of Graphic
    Show Child Window of Single Part List
}
```

HWND Open(HWND hwndParent, LPRECT lpRect);

hwndParent: parent window handler

lpRect: far pointer of rect area

Returns: window handler

Description:

call OpenWindow to create the Damage Page manager with class name "dpgMgr" and style WS_CHILD and WS_BORDER, where hwndParent is an external variable defines in adpdl.h

LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);

hwnd: window handler

wParam: window message word parameter

lParam: window message long parameter

Returns: TRUE means message has processed

Description:

- Save the hwnd in the object

- Load the accelerator table

- Initialize Damage Page manager states

- damageState = damageOff

- drillState = drillOff

- naviState = naviNoIcon

- plistState = plistGraphicOnly

- plistState = plistGraphicOnly

- Initialize the client area and sizes of child window

- Initialize child windows to NULL

- Create 3-line damage list

- Create Navigator without ICON child window

- Create Navigator with ICON child window

- Set Navigator without ICON to be the current child

- Call SetWindowData to set UIC word

- Call MoveChildren to make children visible

LRESULT WMDpgMsg(WPARAM wParam, LPARAM lParam);

wParam: window message word parameter

lParam: window message long parameter

Returns: TRUE means the message has been processed

LRESULT WMAdpMsg(UINT uMsg, WPARAM wParam, LPARAM lParam);

uMsg: window message

wParam: window message word parameter

lParam: window message long parameter

Returns: TRUE means the message has been processed

Description:

switch (message type, uMsg)

{

case ADP_SHOW:

Call MoveChildren to show windows based on the current states

case ADP_HIDE:

HideChildren

case ADP_CLOSE:

Call WMClose function to close Damage Page

case ADP_GRAPHONLYBUTTONCLICKED

Set state plistState to plistGraphicOnly

Call MoveChildren to hide part list and multiple part list, ow graphic window only

case ADP_PANDGBUTTONCLICKED

Set state plistState to plistBoth

if (Part List child window does not exist)
create it

Call MoveChildren to hide multiple part list
show part list and graphic child window

case ADP_PLISTONLYBUTTONCLICKED

Set state to plistPartListOnly

if (Multiple Part List window does not exist)
create it

Call MoveChildren to hide part list and graphics
ow multiple part list

case ADP_ADDLDMGBUTTONCLICKED

To be implemented

case ADP_PLLOADBUTTONCLICKED

To be implemented

case ADP_ZOOMBUTTONCLICKED

if (Graphic Window exists)

Send Zoom message (WM_DPG_MSG,

wParam = ID_dpgZOOM,

lParam = 0) to Graphic Window

case ADP_DAM_AGE LINEBUTTONCLICKED

Toggle the state of damage line damageState

Set the height of damage line window according to
damageState

Announce Message (WM_DPG_MSG,

ID_dpgDAMAGEONOFF, 0)

Call MoveChildren to change window size

case ADP_DRILLBUTTONCLICKED

Toggle drill button state drillState

Announce Message (WM_DPG_MSG,

ID_dpgDRILLBUTTON, 0);

case ADP_VEHICLEBUTTONCLICKED

Toggle navigator state naviState
 Set window size according to the naviState
 Announce Message(WM_DPG_MSG,
 ID_dpgVEHICLEICON, 0)
 Call MoveChildren to change window size

Description:

```

Set cursor to the WAIT cursor
switch (wParam)
{
    case ID_dpgNEWSECTION: // get a new section
        LOWORD(IParam) = section ID
        Update current section
        Route message to child windows and
        let them handle it by AnnounceMessage
    case ID_dpgNEWPART: // get a new part
        LOWORD(IParam) = partID
        Set current part
        switch(action, HIWORD(IParam))
        {
            case ID_peditSHOW:
                Create part edit child window
                by Open
            case ID_peditHIDE:
                Hide part edit child window
        }
        Route Message to child window for handling
    }
    Set cursor back to the original cursor
  
```

LRESULT WMCommand(WPARAM wParam, LPARAM lParam);
 wParam: window message word parameter
 lParam: window message long parameter
 Returns: TRUE means the message has been processed
 if (wParam == ID_dpgCYPHER, ID_dpgHOTSPOT,
 ID_dpgPOLYGON, ID_dpgNOSPOT)
 inform child hot spot location is changed

int TranslateMessage(LPMSG lpMsg);
 lpMsg: far pointer of window message
 Returns: FALSE means the message needs to be translated
 if (Accelerate Table is valid and Translate Message succeeds)
 return TRUE
 if (the message is for one of the dialog windows
 and Translate succeeds)
 return TRUE
 return FALSE

```

};

extern CdpgDamageList dpgDamageList;

#endif // DPGDLIST_H

```

5.2.4 LIBCLASS CdpgNaviIcon: public CdpgWindow

```

// dpgNavi.h
// dpgNavigator class

#ifndef DPGNAVI_H
#define DPGNAVI_H

#include <windows.h>
#include "dpogui.h"

#define segmentMAX 12

// *****
// CdpgNaviIcon
// navigator
//
LIBCLASS CdpgNaviIcon : public CdpgWindow {
private:
    HBITMAP hbmpCar; // navigator car image
    int numSegment; // number of segments
    HBRUSH whitebrush, redbrush; // red and white color hot
spots
    int nCurrentSegment; // currently selected segment
    LRESULT checkGrayDrill(); // grays the drill buttons if
necessary
    short sClicked_x, sClicked_y; // mouse click coordinates

    /* location of drill hotspot on screen */
    POINT GetVehicleHotSpot(int inSeg);
public:

    HWND Open(HWND hwndParent, LPRECT lpRect);

    LRESULT WmCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
    LRESULT WmDestroy(WPARAM wParam, LPARAM lParam);
    LRESULT WmDpgMsg(WPARAM wParam, LPARAM lParam);
    LRESULT WmPaint(WPARAM wParam, LPARAM lParam);
    LRESULT WmCommand(WPARAM wParam, LPARAM lParam);
    LRESULT WmMouse(UINT uMsg, WPARAM wParam, LPARAM lParam);
    LRESULT WmOthers(UINT uMsg, WPARAM wParam, LPARAM lParam);

    /* determine if dropdown list is needed */
    VOID NeedSection(BOOL bNeed = TRUE);
};

extern CdpgNaviIcon dpgNaviIcon;

#endif // DPGNAVI_H

```

5.2.6 LIBCLASS CdpGW : public CdpWindow

```
// dpgGW.h
// CdpGW class
//

#ifndef DPGGW_H
#define DPGGW_H

#include <windows.h>

#include "apidefs.h"

#include "dpggui.h"
#include "dpg2db.h"

#define partMAX 64

/* offset x, y for the "Select a Section" prompt */
#define PROMPT_OFFSET_X 50
#define PROMPT_OFFSET_Y 60

enum PARTSTATE { partSingle, partSingleSelected, partBoth,
                  partLeftSelected, partRightSelected, partBothSelected
};

enum HILIGHTSTATE { NOHILIGHT, HILIGHT, DEHILIGHT };

typedef struct tagPOLYGON {
    LPPOINT pts; // allocate dynamically
    int nPoints;
} POLYGON;

typedef struct tagHOTSPOT {
    POINT pt;
    POINT dogleg;
    POINT arrow;
    POINT hspot;
    int index;
} HOTSPOT;

typedef struct tagSELECTPART {
    int nNewPartIndex; // part number from db
    int nCurrentPart; // upper/lower array index
    int nPreHighLight; // upper/lower array index
} SELECTPART;

typedef struct tagPARTREGION {
    HRGN hrgn; // handle to region
    int nIndex; // part number from db
    int nDisplayIndex; // upper lower array index
} PARTREGION;
```

```

/* constants for arrow head */
#define NUMARROWARRAY 4
#define ARROW_OFFSET_X 4
#define ARROW_OFFSET_Y1 8
#define ARROW_OFFSET_Y2 6

#define HOTSPOTSIZE 14
#define HALF_HOTSPOT HOTSPOTSIZE/2

/* draw states */
enum { DRAW_ALL, DRAW_DAMAGED_AND_CURRENT};

LIBCLASS CdpqGW : public CdpqWindow {
private:
    int nDisplayState;           // hotspots or cypherbars
    int nDrawState;             // hotspots on/off
    HBITMAP hbmpSection;        // section image
    RECT rectSection;           // image size
    HOTSPOT hsUpPart[partMAX];   // upper side hot spots
    HOTSPOT hsLoPart[partMAX];   // lower side hot spots
    int cUpperHotSpot;          // number of upper side hot spots
    int cLowerHotSpot;          // number of lower side hot spots
    SELECTPART selectPart;       // current part
    PARTSTATE partState[partMAX]; // state of parts
    PARTREGION rgnPart[partMAX]; // regions of parts
    POLYGON polygonPart[partMAX]; // polygon of parts
    char currRegionIndex;        // index for current rgn
    char hiRegionIndex;          // index for highest rgn index needed
    because
    not always                  // # of parts and regions index are
                                // in synch
    int numPart;                // number of segments
    BOOL bZoomCur;             // zoom cursor on or not
    HCURSOR hcurZoom;           // zoom cursor
    int cImgShiftX;             // x shift amount
    int cImgShiftY;             // y shift amount

    HBITMAP hbmpBigStatusMoon[TWO_BOTH+1];
    HBITMAP hbmpHiLightStatusMoon[TWO_BOTH+1];

    /* get status bitmap */
    HBITMAP GetStatusMoon(int inPart, HILIGHTSTATE);

    /* position image within window */
    VOID gwAdjustImagePos();

    /* refresh cypher bars */
    VOID gwDrawCyberbar(HDC);

    /* cypher bar arrow */
    VOID gwDrawArrow(HDC hDC, POINT pt, POINT pt1);

    /* draw cypher bar line and circle */
    VOID gwDrawLineNCircle(HDC hDC, int id, HILIGHTSTATE);

    /* de-highlight old part */
    VOID gwDeHighLight();

    /* highlight new part */
    VOID gwHighLight(HDC hDC);

    /* reverse colors for polygon region */

```

```

VOID gwInvertRegion(int iRegion);

/* refresh window for new section */
VOID gwNewSection();

/* set up for new part */
VOID gwNewPart(int PartId);

/* make sure hotspots do not overlap */
VOID gwStaggerHotSpot();

/* test if mouse click within polygon region */
BOOL gWHitTest(int x, int y);

/* draw assembly info in cypher bar */
VOID gwWriteAsm(HDC hDC, int id, int x, int y, HIGHLIGHTSTATE);

/* draw circle portion of cypher bar */
VOID gwDrawSingleCircle(HDC, int, int, BOOL bClearArea = FALSE);

/* cypher bar point */
BOOL GetCyberLoc(int nPart, LPPOINT lpLoc, LPINT bUpperPart);

/* dog leg location */
BOOL GetDogLegLoc(int nPart, LPPOINT lpLoc);

/* arrow head location */
BOOL GetArrowHeadLoc(int nPart, LPPOINT lpLoc);

public:
/* sort comparision routine */
VOID hotspotswap(HOTSPOT& px, HOTSPOT& py);

/* compare hotspots */
int hotspotcmp(HOTSPOT& h1, HOTSPOT& h2);

/* sort hotspots */
VOID dpgsort( HOTSPOT * v, int size);

/* draw hotspot only */
VOID gwDrawHotSpot4Cyber(HDC);

/* set display state to cypher bars or hotspots */
VOID SetCYP_HSPOT(int);

/* set draw state to ALL or DRAW_DAMAGED_AND_CURRENT */
VOID SetDrawState(int);

HWND Open(HWND hwndParent, LPRECT lpRect);
LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
LRESULT WMDestroy(WPARAM wParam, LPARAM lParam);
LRESULT WMSize(WPARAM wParam, LPARAM lParam);
LRESULT WMPaint(WPARAM wParam, LPARAM lParam);
LRESULT WMMouse(UINT uMsg, WPARAM wParam, LPARAM lParam);
LRESULT WMDpgMsg(WPARAM wParam, LPARAM lParam);
LRESULT WMCommand(WPARAM wParam, LPARAM lParam);
LRESULT WMOthers(UINT uMsg, WPARAM wParam, LPARAM lParam);
};

extern CdpGw dpgGW;

#endif // DCPGW_H

```

5.2.8 LIBCLASS CdpGPartList : public CdpGMPartList

```
-----
// dpgPlist.h
// CdpGPartList and CdpGMPartList classes

#ifndef DPGPLIST_H
#define DPGPLIST_H

#include "dpgdefs.h"
#include "dpggui.h"

LIBCLASS CdpGMPartList : public CdpGWindow {
protected:
    int nBottom;    // y coordinate for bottom of list
public:
    HWND Open(HWND hwndParent, LPRECT lpRect);
    LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
    LRESULT WMDestroy(WPARAM wParam, LPARAM lParam);
    LRESULT WMSize(WPARAM wParam, LPARAM lParam);
    LRESULT WMDpgMsg(WPARAM wParam, LPARAM lParam);
    LRESULT WMCommand(WPARAM wParam, LPARAM lParam);
    LRESULT WMOthers(UINT uMsg, WPARAM wParam, LPARAM lParam);
};

LIBCLASS CdpGPartList : public CdpGMPartList {
private:
public:
    HWND Open(HWND hwndParent, LPRECT lpRect);
    LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
    LRESULT WMMouse(UINT uMsg, WPARAM wParam, LPARAM lParam);
    LRESULT WMDpgMsg(WPARAM wParam, LPARAM lParam);
    LRESULT WMCommand(WPARAM wParam, LPARAM lParam);
    LRESULT WMOthers(UINT uMsg, WPARAM wParam, LPARAM lParam);
};

extern CdpGMPartList dpgMPartList;
extern CdpGPartList dpgPartList;

#endif // DPGPLIST_H
-----
```

5.2.9 LIBCLASS CdpGPartEdit : public CdpGWindow

```

// dpgPedit.h
// CdpGPartEdit class

#ifndef DPGPEDIT_H
#define DPGPEDIT_H

#include <windows.h>
#include "apidefs.h"
#include "dpggui.h"
#include "dpggrid.h"

#define NO_OF_PARTEDIT 5
#define THREE_PIXELS 3
#define TWO_PIXELS 2
#define BUTTONS_OF_PARTEDIT 5

#define MAXPARTNAME 40
#define MAXPARTNUMBER 40

/* constants for control positions
in unit of pixels */

/* grid */
#define PE_GRID_HEIGHT 288
#define PE_HALF_GRID_HEIGHT 56
#define PE_GRID_X 8
#define PE_GRID_Y 47

/* part name and part number */
#define PE_PARTNAME_X 5
#define PE_PARTNAME_Y 2
#define PE_PARTNO_X 350
#define PE_PARTNO_Y 2
#define PE_PARTNAMEWIDTH 320
#define PE_TEXTHEIGHT 20

/* buttons */
#define PE_BUTTONWIDTH 73
#define PE_ALLOP_BUTTONWIDTH 100
#define PE_BUTTONHEIGHT 23
#define PE_HALF_BUTTONY 124

#define PE_OK_X 4
#define PE_CANCEL_X (PE_OK_X + PE_BUTTONWIDTH + 2)
#define PE_ADDDEFAULT_X (PE_CANCEL_X + PE_BUTTONWIDTH + 2)
#define PE_RI_X (PE_ADDDEFAULT_X + PE_BUTTONWIDTH + 2)
#define PE_ASSEMBLY_X (PE_RI_X + PE_BUTTONWIDTH + 2)
#define PE_PARTIAL_X (PE_ASSEMBLY_X + PE_BUTTONWIDTH + 2)
#define PE_HELP_X (PE_PARTIAL_X + PE_ALLOP_BUTTONWIDTH + 2)

```

```

/* part edit dialog */
#define PE_DLG_X -24
#define PE_DLG_Y -60
#define PE_DLG_WIDTH 560
#define PE_DLG_HEIGHT 195
#define PE_DLG_HEIGHT_OFFSET 20

#define MAXCONTROLS 10

LIBCLASS CdpGPartEdit : public CdpGWindow {
private:
    HWND hWndControl[MAXCONTROLS];
    int cControls; // number of child controls ( not include
grid )
    int nCurrentControl; // 0, 1, 2 ...

    /* override the Register in CdpGWindow */
    ATOM Register(LPSTR lpClass);

    /* init choices, and assembly */
    VOID InitAssembly();
    VOID InitDefault();
    VOID CreateControls();

    /* display current choice info */
    void ShowChoiceInfo(HDC hDC);

    /* keyboard navigation */
    int PrevControl()
    { return nCurrentControl = (nCurrentControl <= 0) ?
cControls-1 : nCurrentControl-1; }
    int NextControl()
    { return nCurrentControl=(nCurrentControl >= cControls-1)
? 0 : nCurrentControl+1; }

    /* get id of control */
    int ControlID(int n);

protected:
    BOOL bFullSize; // collapse/full

    CdpGGrid * pGrid;
    CdpGPart * pCurPart;

    /* change part edit size */
    VOID ExtentWindow();

    /* default pushbutton */
    void SetDefaultButton(int nID);

    /* enable/disable add r&l button */
    VOID InitRI();

    /* print title for grid */
    VOID PrintColumnTitles(HDC hDC, int y );

public:
    CdpGPartEdit(CdpGPart * pThisPart) { pGrid = new
CdpGPartGrid(pCurPart = pThisPart); }
    CdpGPartEdit(void) { if (pGrid) delete pGrid; }

    VOID MoveTab();
    VOID DoReturn();

```

```

    BOOL IsPEditFullSize() { return bFullSize; };

    HWND Open(HWND hwndParent, LPRECT lpRect);

    virtual LRESULT WMCreat(HWND hwnd, WPARAM wParam, LPARAM lParam);
    virtual LRESULT WMClose(WPARAM wParam, LPARAM lParam);
    virtual LRESULT WMPaint(WPARAM wParam, LPARAM lParam);
    virtual LRESULT WMCommand(WPARAM wParam, LPARAM lParam);
    virtual LRESULT WMOthers(UINT uMsg, WPARAM wParam, LPARAM lParam);
    virtual LRESULT WMDpgMsg(WPARAM wParam, LPARAM lParam);

    /* keyboard */
    HWND GetGridWnd() { return pGrid->GetWnd(); }
    BOOL ProcessKey(WPARAM wParam, LPARAM lParam)
    { return pGrid->ProcessKey(wParam, lParam); }
};

LIBCLASS CdpgMCodePartEdit : public CdpgPartEdit {
public:
    CdpgMCodePartEdit(CdpgPart * pThisPart) : CdpgPartEdit(pThisPart) {
        pGrid = new CdpgMCodeGrid(pCurPart = pThisPart);
    }
    LRESULT WMCommand(WPARAM wParam, LPARAM lParam);
    LRESULT WMDpgMsg(WPARAM wParam, LPARAM lParam);
};

extern CdpgPartEdit dpgPartEdit;
extern CdpgMCodePartEdit dpgMCodePartEdit;

#endif // DPGPEDIT_H

```

5.2.10 LIBCLASS CdpGmfwin : public CdpGWindow

```
// CdpGmfwin class for Multi Function Window
// 1. A window with vertical scroll bar
// 2. It has row element which is defined in CdpGDListRow
// 3. The CdpGmfwin is registered and appeared in the damage list
// dialog as the same role as other child control.

#ifndef DPGMFWIN_H
#define DPGMFWIN_H

#include <windows.h>

#include "dpggui.h"
#include "dpgrow.h" // for rows
#include "dpg2db.h"

//
// define constants
//

// constants for scrolling
enum { THREELINE, SEVENTEENLINE, ONEMOREFOR3, ONEMOREFOR17 };
enum tagNextMove { cnNextRow, qnNextCol, cnNextStay };

#define MAX_GUIDE 25

typedef struct tagINPUTITEM {
    char szText[32];
    int nMaxLen;
} INPUTITEM, *PINPUTITEM, FAR * LPINPUTITEM;
```

```

LIBCLASS CdpGmfwIn : public CdpGWindow {
protected:
    // data members
    HWND hWnd;

    CdpGDListRow Rows;

    int nStartDmg;           // the first damage on the display
    int nTotalDmg;           // total number of damages from db

    // current pointers, for keyboard interface
    int nDeltaRow;           // one page size
    int nCurrentRow;         // on screen
    int nCurrentCol;

    INPUTITEM inputItems[10];

    // flags
    int nViewAll;           // view all / view open

    // draw functions
    VOID DrawDamage(HDC hdc, int nR, int nDmg);
    VOID DrawInputItems(HDC hdc);
    VOID DrawClear(HDC hdc, int nR);
    VOID DrawGrid(HDC);

    // current item functions
    int SetCurrentRow(int nR);
    int SetCurrentCol(int nC);
    int SetStartDmg(int nR);
    int SetCurrentItem(int nR, int nC);

    // keyboard input functions
    void InitInputItems();
    BOOL IsGoodChar(int nChar);
    BOOL DeleteAChar(int nR, int nC);
    BOOL AppendAChar(int nR, int nC, int nChar);

    // input complete
    int IsGoodGuideNumber();
    int IsGoodMCode();
    int IsGoodManual();
    int IsGoodOpCode();
    BOOL AddGuideDamage();
    BOOL AddMCodeDamage();
    BOOL AddManualDamage();
    int CompleteItem(int nR, int nC);
    BOOL ToPartEdit();

public:
    CdpGmfwIn() {};
    ~CdpGmfwIn() {};

    HWND Open(HWND hwndParent, LPRECT lpRect);

    HWND GethWnd() { return hWnd; }

    int ReGetTotalDmgRecord()
    { return nTotalDmg = dpgDamageServer.GetDmgRecordNumber(nViewAll); }

    int SetRowHeight(int type)
    { nDeltaRow = (type == WIDEFORMAT) ? 17 : 3;
      return Rows.SetHeight(type); }

```

```

int ReSetScroll();

int GetRowHeight()
{ return Rows.GetHeight(); }

int GetCurrentRow() { return nCurrentRow; }
int GetCurrentCol() { return nCurrentCol; }

int col2x(int nC) { return Rows.col2x(nC); }
int row2y(int nR) { return Rows.row2y(nR); }
int GetCols() { return Rows.GetCols(); }

VOID Hilight(int nR, int nC) { Rows.Hilight(hWnd, nR, nC); }
VOID Outline(int nR, int nC, BOOL bShow = TRUE) { Rows.Outline(hWnd,
nR, nC, bShow); }

// message processing functions
LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
LRESULT WMMouse(UINT uMsg, WPARAM wParam, LPARAM lParam);
LRESULT WMKey(UINT uMsg, WPARAM wParam, LPARAM lParam);
LRESULT WMChar(UINT uMsg, WPARAM wParam, LPARAM lParam);
LRESULT WMSize(WPARAM wParam, LPARAM lParam);
LRESULT WMPaint(WPARAM wParam, LPARAM lParam);
LRESULT WMScroll(UINT uMsg, WPARAM wParam, LPARAM lParam);
LRESULT WMCommand(WPARAM wParam, LPARAM lParam) {return TRUE;}
LRESULT WMGetDlgCode();
LRESULT WMDpgMsg(WPARAM wParam, LPARAM lParam);
LRESULT WMOthers(UINT uMsg, WPARAM wParam, LPARAM lParam);
};

#endif // DPGMFWIN_H

```

5.2.11 LIBCLASS CdpgLineEdit: public CdpgWindow

```
// dpgDlgLE.h
// Model dialog boxes
//

#ifndef DPGDLGLE_H
#define DPGDLGLE_H

#include <windows.h>
#include "apidefs.h"

#include "dpggui.h"
#include "dpg2db.h"

// return status
enum { cnLINEEDIT_CANCEL, cnLINEEDIT_MODIFY, cnLINEEDIT_PEDIT,
      cnLINEEDIT_GRAPHIC, cnLINEEDIT_DELETE, cnLINEEDIT_INC};

// *****
// CdpgLineEdit
//
LIBCLASS CdpgLineEdit : public CdpgWindow {
protected:
    char szDesc[24];
    POINT ptStart;          // start location
    int nValue;             // value of button chosen

    virtual LRESULT Dook() { return FALSE; } // if user selects
enter
public:

    CdpgLineEdit(LPPOINT ipt) {ptStart = *ipt; };

    HWND Open(HWND hwndParent, LPRECT lpRect);
    LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
    LRESULT WMCommand(WPARAM wParam, LPARAM lParam);

    /* get operation which was chosen */
    int GetValue() { return nValue; }

private:
};

// *****
```

```

// CdpGIncOps
//
LIBCLASS CdpGIncOps : public CdpGGeneralList {
private:
    char szDesc[24];
    POINT ptStart;           // start location
    DAMAGEENTRY de;          // current dmg record

public:
    CdpGIncOps(LPSTR lpszValue, POINT ipt, LPDAMAGEENTRY lpDE, int
    ichanged);
    LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
    LRESULT WMSize(WPARAM wParam, LPARAM lParam);
};

/* function to be called by engine enum function */
BOOL _export _far IncOpCallback(int nestLevel, LPSTR szDesc, DWORD
lData);

#endif // DPGDLGLE_H

```

5.2.13 LIBCLASS CdpGCalculator : public CdpGWindow
 5.2.14 LIBCLASS CdpGCalPrice : public CdpGCalculator
 5.2.15 LIBCLASS CdpGCalAdj : public CdpGCalculator
 5.2.16 LIBCLASS CdpGCalHour : public CdpGCalculator

```
// dpgDlg.h
// Model dialog boxes
//
#ifndef DPGDLG_H
#define DPGDLG_H

#include <windows.h>
#include "apidefs.h"

#include "dpggui.h"
#include "dpg2db.h"

// Defines
#define ID_CAL_MARKUP 0
#define ID_CAL_DISCOUNT 1
#define ID_CAL_BETTERMENT 2
#define ID_CAL_ADDTO 3
#define ID_CAL_OVERRIDE 4

#define MAX_CAPTION_LEN 80

// ***** Calculators Section *****
// -----
// CdpGCalculator, the base class for the calculators
//
LIBCLASS CdpGCalculator : public CdpGWindow {
protected:
    DAMAGEENTRY de; // de record to modify
    int flag; // status
    char szDesc[24]; // header caption
    char szValue[10]; // caption value

    char szCaption[MAX_CAPTION_LEN + 1];
    POINT ptStart; // start location
    HBRUSH newbrush; // newbrush to paint the caption bar

    RECT rectCalc; // rect for the calculator
    POINT ptCap; // caption start location
    int iCapWidth; // caption width
    int iCapHeight; // caption height
    BOOL bCapLock; // cap lock
    BOOL bChanged; // modified?

    virtual LRESULT Dook(); // if user selects enter

    // to draw text on the caption button
    VOID DrawTextOnCap(BOOL bFull);
    VOID Draw3DLine(HDC hdc, BOOL bWhite);
    BOOL IsHit(LPARAM);

public:
    CdpGCalculator(DAMAGEENTRY *lpDE, POINT ipt);

    HWND Open(HWND hwndParent, LPRECT lpRect);
    LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
    LRESULT WMCommand(WPARAM wParam, LPARAM lParam);
};
```

```

LRESULT WMOthers(UINT uMsg, WPARAM wParam, LPARAM lParam);
LRESULT WMDestroy(WPARAM wParam, LPARAM lParam);

/* retrieve current value and type flag */
void GetValue(LPSTR lpszResult, LPINT lpflag);

private:
/* add next digit entered by user */
LRESULT dpgAddDigit(char *);
};

// *****
// CdpGCalPrice
//
LIBCLASS CdpGCalPrice : public CdpGCalculator {
public:
    CdpGCalPrice(DAMAGEENTRY *lpDE, POINT ipt)
        : CdpGCalculator(lpDE, ipt) {}

    HWND Open(HWND hwndParent, LPRECT lpRect);
    LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
    LRESULT DoOK(); // if user selects enter
};

// *****
// CdpGCalAdj
//
LIBCLASS CdpGCalAdj : public CdpGCalculator {
public:
    CdpGCalAdj(DAMAGEENTRY *lpDE, POINT ipt)
        : CdpGCalculator(lpDE, ipt) {}

    HWND Open(HWND hwndParent, LPRECT lpRect);
    LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
    LRESULT DoOK(); // if user selects enter
};

// *****
// CdpGCalHour
//
LIBCLASS CdpGCalHour : public CdpGCalculator {
public:
    CdpGCalHour(DAMAGEENTRY *lpDE, POINT ipt)
        : CdpGCalculator(lpDE, ipt) {}

    HWND Open(HWND hwndParent, LPRECT lpRect);
    LRESULT WMCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
    LRESULT DoOK(); // if user selects enter
};

```

```

// ***** Other dialog boxes *****
// -----
// CdpgRates
//
LIBCLASS CdpgRates : public CdpgGeneralList {
private:
    char rateCode[3];
    POINT ptStart;
    int changed;

public:
    CdpgRates(LPSTR lpszValue, POINT ipt, int ichanged);
    LRESULT WmCreate(HWND hwnd, WPARAM wParam, LPARAM lParam);
    /* retrieve current value and type flag */
    void GetValue(LPSTR lpszRateCode, LPINT lpchanged);
};
#endif // DPGDLG_H

```

5.3 Database Interface Objects

5.3.1 LIBCLASS CdpvgVehicle

```
// dpgDBveh.h
// Vehicle database and data access classes
//
#ifndef DPGDBVEH_H
#define DPGDBVEH_H

#include <windows.h>
#include "apidefs.h"

#include "dpgdefs.h"

// *****
// CdpvgVehicle
// vehicle data such as sections, vehicle icon, drill and so on
//

LIBCLASS CdpvgVehicle {
protected:
    HBITMAP bmpVehicle; // vehicle navigator image
    int nCurSec; // current section
    int nCurAddImg; // additional image index
    int nCurDrill; // offset to current drill
    int nCurOrientation; // current orientation
    BOOL bLoad[16]; // loaded drill sections

public:
    CdpvgVehicle();
    ~CdpvgVehicle();

    /* query functions */
    int GetCurSection() { return nCurSec; }
    int GetCurAddlImg() { return nCurAddImg; }

    /* open and close database */
    BOOL OpenVehicle();
    BOOL CloseVehicle();

    /* orientation */
    int GetOrientation() { return nCurOrientation; }
    int SetOrientation(int inOri);

    /* navigator car image */
    HBITMAP GetVehicleImage() { return bmpVehicle; }

    /* number of drill sequences */
    int GetDrillSequenceCount();

    /* points for vehicle icon hotspots */
    POINT GetVehicleHotSpot(int inSeg);

    /* segment for drilling */
    int FillDrillSequenceList(HWND hwnd, int nSeg);
    int GetDrillSegment(int inSec);
    int FindSectionFromDrill(int inSeg);
    int FirstSectionFromDrill(int inSeg);
}
```

```

int NextDrillSection();
int PrevDrillSection();

/* sections list */
void FillSectionsList(HWND hwnd);

/* additional images for a section */
int FillAddlImages(HWND hwnd, int inSec);

/* current section */
int GetCurrentSection() { return nCurSec; }
int SetCurrentSection(int inSec);
int SetCurrentNode(int inSec, int inAddImg = 0);

/* multi-section parts list */
BOOL IsAreaLoaded();
void FillLoadAreas(HWND hwnd);
void SetLoadedAreas(HWND hwnd);
int FillMultiPartsList(HWND hwnd, BOOL bResetContents = FALSE);
BOOL DrawMultiPartsListItem(LPDRAWITEMSTRUCT lpDis, int nBottom);

/* additional damages */
void FillStandard(HWND hwnd);
void FillAdditionalLabor(HWND hwnd);
void FillInspect(HWND hwnd);

/* other information */
BOOL GetSectionDesc(int inSec, LPSTR szDesc);
BOOL GetNodeDesc(int inSec, int inAddImg, LPSTR szDesc);
BOOL GetCurrentNodeDesc(LPSTR szDesc);
};

// vehicle object
extern CdpgVehicle dpgVehicle;

#endif // DPGDBVEH_H

```

5.3.2 LIBCLASS CdpGCurSection

```
// dpgDBsec.h
// Current section database and data access classes
//
#ifdef DPGDBSEC_H
#define DPGDBSEC_H

#include <windows.h>
#include "apidefs.h"
#include "dpg2db.h"

// *****
// CdpGCurSection
// current section for graphic and part list
// What is the relation between CdpGVehicle and CdpGCurSection ???

/* image size in pixels */
#define cnpBIGIMAGEX 384
#define cnpSMALLIMAGEX 320

#define MAXPOLYPTS 64
#define MAXPARTS 64

// status moon array
#define SideMASK (1 << 0)
#define LeftDamageMASK (1 << 1)
#define RightDamageMASK (1 << 2)

enum statusIndex { ONE_NO, TWO_NO, ONE_LEFT, TWO_LEFT,
                  ONE_RIGHT, TWO_RIGHT, ONE_BOTH, TWO_BOTH };

class CdpGCurSection {
protected:
    HBITMAP hbmpStatusMoon[TWO_BOTH + 1]; // size of image in pixel
    RECT rcImage; // section image
    HBITMAP hbmpBigImage;
    HBITMAP hbmpSmallImage;
    int numParts; // number of parts
    int nCurPart; // current part

    /* init for new section */
    void ClearSection();

public:
    CdpGCurSection();
    ~CdpGCurSection();

    /* clear section and part */
    BOOL NewNode();

    /* query function big/small image */
    BOOL IsBigImage() { return (rcImage.right == cnpBIGIMAGEX) ? TRUE :
FALSE; };

    /* image size */
    BOOL SetImageRect(LPRECT lpRect);
    BOOL GetImageRect(LPRECT lpRect);
}
```

```

/* section image */
HBITMAP GetImage(HDC hdc);
HBITMAP GetZoomImage(HDC hdc, int inPart) { return NULL; }

/* parts or hot spots */
int GetNumberOfParts() { return numParts; }

/* description text */
BOOL GetPartDesc(int inPart, LPSTR lpDesc);

/* determine if part is damaged */
int GetPartStatus(int inPart);

/* status moon for display */
HBITMAP GetStatusMoon(int inPart);

/* status moon for part choice display */
HBITMAP GetBMPMoon(int curStatus);

/* cypher bar point */
BOOL GetCyberLoc(int inPart, LPPOINT lpLoc);

/* dog leg location */
BOOL GetDogLegLoc(int inPart, LPPOINT lpLoc);

/* arrow head location */
BOOL GetArrowHeadLoc(int inPart, LPPOINT lpLoc);

/* get polygon points for region setup */
BOOL GetPartPolygon(int inPart, LPPOINT lpPoly, LPINT nPt, int shiftX
                    = 0, int shiftY = 0);

/* get part region for hittest */
HRGN GetPartRegion(int inPart, int shiftX = 0, int shiftY = 0);

/* get z-order for region overlaps */
int GetPartRegionOrder(int inPart);

/* single section parts list */
int FillPartsList(HWND hwnd, BOOL bResetContents = FALSE);

/* owner-draw parts list */
BOOL DrawPartsListItem(LPDRAWITEMSTRUCT lpDis, int nBottom);

/* current part processing */
void SetCurrentPart(int inPart);
int GetCurrentPart() { return nCurPart; }
BOOL IsItSinglePart();

/* query whether opcode in the current section */
BOOL IsGuideIn(LPDAMAGEENTRY lpDe, LPINT lpPart, LPINT lpChoice);
BOOL IsSameBranch(int inPart1, int inComparePart = (-1));
}

// current section object
extern CdpgCurSection dpgCurSection;

#endif // DPGDBSEC_H

```

5.3.3 LIBCLASS CdpGCurPart

```
// dpgDBpar.h
// Current part database and data access classes
//
#ifndef DPGDBPAR_H
#define DPGDBPAR_H

#include <windows.h>
#include "apidefs.h"
#include "dpg2db.h"

// *****
// CdpGCurPart
// current part for part edit; current coordinate would be more
// proper.
//
// What is the relation between CdpGCurPart and CdpGCurSection?
//

typedef struct tagOPCODEDATA {
    int nCurCheck;
    int nCheck;
    BOOL bDirty;
    DAMAGEENTRY dr;
} OPCODEDATA, * PPCODEDATA, FAR * LPCODEDATA;

LIBCLASS CdpGPart {
private:
protected:
    CdpGPart() {}

    /* record modified; passive dirty bit */
    BOOL bDirty;

    int nCurPart;

    /* orientation */
    int nCurOri;

    /* part choice names list */
    int numChoices;           // number of choices
    int nCurChoice;          // current choice

    virtual int SetCurChoice(int inChoice) { return inChoice; }

    /* make sure only one replace op is checked */
    int SetReplaceStatus(int inOp, int inOri, int inStatus);

    /* opcode information for current choice */
    OPCODEDATA opData[cnBoth][OP_MAXSIZE];

    /* replace opcodes for left and/or right */
    int nCurReplace[cnBoth];
    int numReplaces;

    /* remove all damaged ops */
    BOOL DeleteAll(int inOri);

    /* init new part */
}
```

```

void Clear();

/* set dirty bit for one opcode */
void SetDirty(int inOp);

public:
    BOOL GetDirty() { return bDirty; }
    void ForceSetDirty(BOOL ibDirty) { bDirty = ibDirty; }

    virtual int GetReplaceOpIndex(int nth) { return (-1); }
    virtual int IsReplaceOp(int inOp);

    virtual int GetNumOtherOps() { return 0; }
    virtual int GetOtherOpIndex(int nth) { return 0; }

    /* get opcode data of current choice */
    virtual BOOL GetOpDesc(int inOp, LPSTR lpOpDesc) { return 0; }
    BOOL GetPrice(int inOp, LPSTR lpPrice, LPINT lpnFlag);
    BOOL GetAdjust(int inOp, LPSTR lpAdjust, LPINT lpnFlag);
    BOOL GetHour(int inOp, LPSTR lpHour, LPINT lpnFlag);
    BOOL GetRate(int inOp, LPSTR lpRate, LPINT lpnFlag);
    BOOL GetTotal(int inOp, LPSTR lpTotal);
    BOOL GetOpen(int inOp);

    /* base data */
    BOOL GetBasePrice(int inOp, LPSTR lpPrice);
    BOOL GetBaseHour(int inOp, LPSTR lpHour);
    BOOL GetBaseRate(int inOp, LPSTR lpRate);
    BOOL GetBaseAdjust(int inOp, LPSTR lpAdjust);

    /* fill damage record with opcode values */
    BOOL GetDmgRecord(int inOp, LPDAMAGEENTRY lpDe);

    /* set opcode data of current choice */
    BOOL SetPrice(int inOp, LPSTR lpPrice, int nFlag = PRICE_OVERRIDE);
    BOOL SetAdjust(int inOp, LPSTR lpAdjust, int nFlag = DISCOUNT);
    BOOL SetHour(int inOp, LPSTR lpHour, int nFlag = HOURS_OVERRIDE);
    BOOL SetRate(int inOp, LPSTR lpRate, int nFlag = RATES_OVERRIDE);
    BOOL SetOpen(int inOp, BOOL ibOpen);

    /* set opcode data of current choice */
    int GetStatus(int inOp, int iOri);
    int SetStatus(int inOp, int inOri, int inStatus);
    int GetOrientation() { return nCurOri; }

    /* orientation */
    virtual BOOL IsSingle() { return TRUE; }
    void SetOrientation(int inOri);

    /* replacement info */
    virtual int GetNumReplaces() { return numReplaces; }
    virtual int GetCurReplace() { return nCurReplace[nCurOri]; }

    /* get opcode base values */
    virtual BOOL GetBaseValues(int inOp, LPDAMAGEENTRY lpDe) { return 0; }
    virtual BOOL GetFirstAvailOpDE(LPDAMAGEENTRY lpDe) { return 0; }

    /* query and add changes to database */
    virtual void UpdateMessage() { return; }
    virtual BOOL UpdateChange(int inOri);
    virtual BOOL UpdateChanges();

```

```

virtual BOOL IsDamaged() { return FALSE; }
virtual BOOL GetCurChoiceDesc(LPSTR lpDesc) { return FALSE; }
virtual BOOL GetPartNumber(LPSTR lpNum) { return FALSE; }

BOOL IsPartOfAssembly(LPDAMAGEENTRY);

/* refinish breakout */
BOOL GetRBKSurfaceHours(int inOp, LPSTR lpSurface);
BOOL GetRBKEdgeHours(int inOp, LPSTR lpEdge);
BOOL GetRBKTwoStageHours(int inOp, LPSTR lpTwoStage);
BOOL GetRBKTwoStageSetupHours(int inOp, LPSTR lpTwoStageSetup);
BOOL GetRBKAdjHours(int inOp, LPSTR lpAdj);
};

```

```

LIBCLASS CdpGCurPart : public CdpGPart{
private:
    BOOL AddEnL(int inOri);
public:
    CdpGCurPart();
    ~CdpGCurPart() { Clear(); }

    /* set up for new part */
    BOOL NewPart(int inPart);

    /* orientation */
    BOOL IsSingle();

    /* part choices */
    int GetNumChoices() { return numChoices; }
    BOOL FillChoices(HWND hwnd, BOOL bResetContents = FALSE);

    /* current part choice */
    int SetCurChoice(int inChoice);
    int GetCurChoice() { return nCurChoice; }
    BOOL GetCurChoiceDesc(LPSTR lpDesc);
    BOOL GetPartNumber(LPSTR lpNum);
    BOOL GetGuideNumber(LPSTR lpGuide);
    BOOL IsDamaged();

    /* replacement info */
    int GetNumReplaces() { return numReplaces; }
    int GetReplaceOpIndex(int nth);

    /* other opcodes info */
    int GetNumOtherOps();
    int GetOtherOpIndex(int nth);

    /* get opcode data of current choice */
    BOOL GetOpDesc(int inOp, LPSTR lpOpDesc);

    /* get opcode base values */
    BOOL GetBaseValues(int inOp, LPDAMAGEENTRY lpDe);
    BOOL GetFirstAvailOpDE(LPDAMAGEENTRY lpDe);

    /* add E and L if they exist */
    BOOL AddDefault();

    /* try to add r&l */
    BOOL AddRI(void);

    /* assembly tree for current part */

```

```

int FillAssembliesList(HWND hwnd);
int GetPartAsm(int inPart);
char GetPartAsmType( int inPart);

/* query and add changes to database */
void UpdateMessage();

/* Owner draw part choice */
BOOL DrawPartsChoiceItem(LPDRAWITEMSTRUCT lpDis);
HBITMAP GetStatusMoon(int inChoice);
};

```

```

LIBCLASS CdpGCurMCode : public CdpGPart{
private:
/* attempt to add E and L as default */
BOOL AddEnL(void);
public:

CdpGCurMCode();
~CdpGCurMCode() { Clear(); }

/* set up for new mcode */
BOOL NewMCode(int inMCode);

BOOL IsDamaged() { return 0; }

/* replacement info */
int GetNumReplaces() { return numReplaces; };
int GetReplaceOpIndex(int nth);

/* other opcodes info */
int GetNumOtherOps();
int GetOtherOpIndex(int nth);

/* get opcode data for current MCode */
BOOL GetOpDesc(int inOp, LPSTR lpOpDesc);

/* set opcode data for current choice */
int SetStatus(int inOp, int inStatus);

/* get opcode base values */
BOOL GetBaseValues(int inOp, LPDAMAGEENTRY lpDe);

/* add E and L if they exist */
BOOL AddDefault();

/* try to add r&i */
BOOL AddRI(void);

/* query and add changes to database */
void UpdateMessage() { return; }
};

// current part object
extern CdpGCurPart dpgCurPart;

// current MCode object
extern CdpGCurMCode dpgCurMCode;

#endif // DPGDBPAR_H

```